Proc. of the Second Intl. Conf. on Advances in Electronics, Electrical and Computer Engineering -- EEC 2013

ISBN: 978-981-07-6935-2

Automated Comparison Framework for Regression Testing

Aastha Kaur, Student, M.Tech , Navdeep Singh,B.Tech

Abstract: **To reduce the effort, testing cycle time & % of human errors that can easily creep in while comparing the results of Regression Test Suite, a thought process was put into designing & implementing an Automation Framework for the purpose. A lot of work and research has already being done for the Execution phase of Regression Testing wherein two parallel sides – Test & Prod are setup & Test Cases executed by firing the same one after the another & results stored. A large number of Regression Automation Tools are available in market like, QTP, Selenium, WATIR etc, to cover this up. Contrary to this very less work is available & very less has been thought about the Comparison phase wherein Test Results thus generated have to be compared to produce a summary report for QA Testers to analyze which they can further categorize into Expected & Unexpected Breaks & then reach out to Development for investigation & thus complete the end-to-end life cycle of Regression Testing. With advent of IT and shift of focus toward Financial Banks & Institutions, a need is felt to have some faster & feasible way to compare records with high volume. That is the starting point for this paper under which an Automation Framework for Comparison Phase of Regression Testing is built in Perl, that could easily cover records of any volume. Use of Industry Compliant Methodology, named Best Match, made the framework even more flexible for scenarios having duplicate records on either of the two parallel sides. Best practice Data Structures like Hash are being used in the implementation that have fasten up the parsing & key pattern filtering, hence lowering down the overall comparison & summary generation time. Use of programming language Perl has made the framework platform or operating system independent as the implementation code can easily be run on any OS, like Unix, Sun, Windows.**

Aastha Kaur
Lingayas University
India

Navdeep Singh
Punjab Technical University
India

Comparing results of Regression Test Suite is far more complex than it seems. The below paper aims toward designing and implementing a framework that could simplify this complexity.

I. Introduction

Regression Testing aims toward testing a piece of code again and again to ensure that new piece of code has not impacted the existing piece. As per previous study[4], this is an expensive process. This testing is performed at various levels of software development life cycle. Right from the start a piece of code is delivered by a developer till the time its go to production, a software tester keeps on testing a code regressively and keep on finding the breaks and get the code improved or fixed. Regression Testing over the period has evolved from being manual to automated. A large number of regression testing automation tools are available in market that a QA tester can use for his/her purpose. A study of the same[2] proves that these tools not only simplify the execution process but also gives a huge gain in the form of less effort, reduce testing cycle time and ease of use[8]. Regression testing in an end-to-end form consists of two basic sub processes 1) **Execution**: The execution part of Regression Testing consists of creating a Regression Test Environment wherein test cases can be grouped and executed to get the test results. Regression Test Environment[1] consists of two sides: a) Test side: Test side consists of what all a software code is available in production plus new code that has been delivered by a developer. This is a new piece of code that will be tested regressively over different testing life cycle stages before it goes into production b) Prod side: This side consists what all a software code is available in production. The need to have this side is to test the new piece of code above the existing piece and get the breaks discovered, analyzed and fixed. A High Level Architectural Diagram showing the two sides as explained above is as below:

Proc. of the Second Intl. Conf. on Advances in Electronics, Electrical and Computer Engineering -- EEC 2013

ISBN: 978-981-07-6935-2

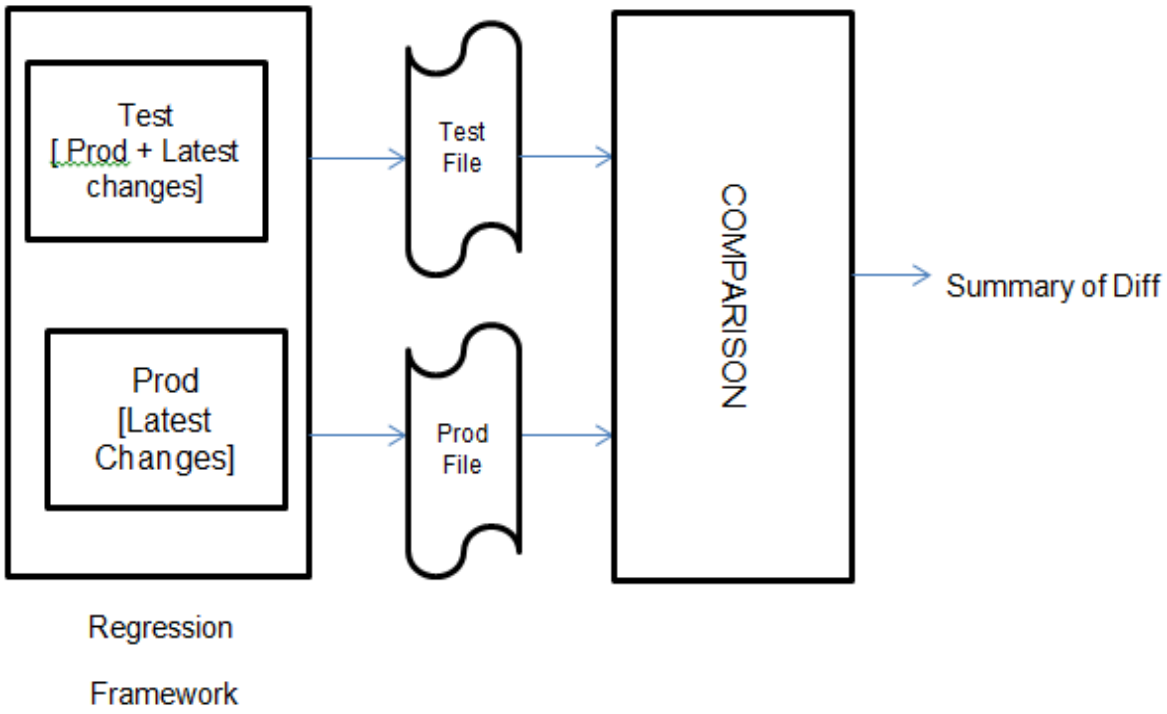

Fig 1: Regression Testing Flow Diagram

Automation tools like QTP serves as a powerful tool[2] to cover all the above functions. Any tester can get into QTP, pick up test cases for a release, setup the Test & Prod Env, use the previous results as baseline and rerun the test cases with the same set of test data to get the next baseline results. The results are in the form of breaks which when analyzed helps to verify that new piece of code has not impacted the existing code. This gives way to next phase of Regression Testing that is Comparison.

2) **Comparison:** Once the Execution Phase gets over, the baseline results are compared to give summary output of differences observed in the columns values of Test and Prod side. Most Regression Test Environments compare the behavior of two program versions to find out if there are any changes. Deviations in the program behavior can be intended, such as bug fixes, or unintended, such as regression faults[3].
Testing Teams over the period of time have found Comparison Activity far more complex than it used to be earlier. Only rationale behind this was the increase in volume of the records generated from the Automated Regression tools. Comparing a small volume of 2-3 records in each of Test & Prod side is far more simpler and straightforward when compared to a volume of 10-15K of records. Very less work has been done or is available in market that could ease off this complexity.

II. Comparison Considerations

When comparing results of Regression Suites, due consideration should be given to volume of data put in for comparison. With advancements coming in the IT industry and increase of client base, the number of Test Records were bound to increase. Manually comparing such a high client base is far more difficult than said. Not only the QA tester has to compare each and every column value of test & prod side one by one but also has to follow some special algorithms/methodologies in case of duplicate records observed. All this only adds to the total testing cycle time with huge effort on the parts of QA testers plus high chances of errors creeping in due to total manual intervention involved. To ease off the manual complexity, a thought process was put into designing and implementation of an Automation Framework that can easily compare Test Records of any volume. Add to this, the Framework can easily handle the duplicacy scenario. A framework consists of a common code that

Proc. of the Second Intl. Conf. on Advances in Electronics, Electrical and Computer Engineering -- EEC 2013

ISBN: 978-981-07-6935-2

provides solutions for several similar applications for specific problem types. Frameworks differ from software libraries, among other things, in two ways: First, the flow of control isn't dictated by the caller, but by the framework (inversion of control). Second, a user can extend a framework by overriding functionality or by implementing interfaces[6].

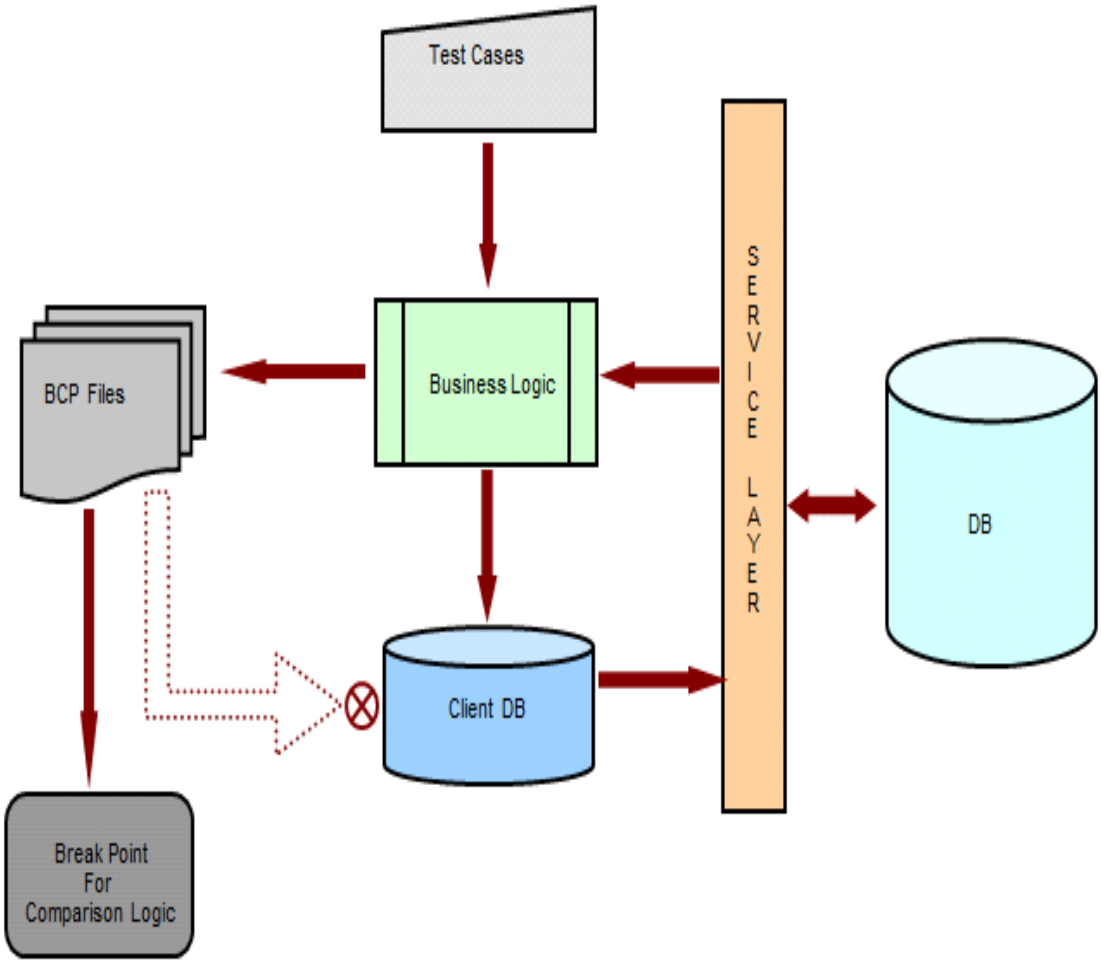

Fig 2: High level Architecture Diagram of Automated Comparison Framework

As shown in Fig 2, the Automation Framework will take as an input the **Data Files** from Test and Prod side which will be passed over to core Business Logic that will cover all processing part. Business Logic will do processing in two forms: a) Non-Duplicate records– It will parse the records to get individual column values which will be compared for equality and differences recorded b) Duplicate records- It will hand over the request to a **Service Layer** that encapsulate the Best Match Methodology. Service layer will return back the Best Pair to the Business Logic which will compare the records the same way as a non-duplicate record was compared. Business logic also hits the **Client DB** to get the Configuration Information like the location from where the Test and Prod input files will be picked up along with location where it will record the differences. The output will be in the form of BCP [Bulk Control Process] files which can either be send over to QA testers over e-mail for further analysis or can be saved in the

Proc. of the Second Intl. Conf. on Advances in Electronics, Electrical and Computer Engineering -- EEC 2013

ISBN: 978-981-07-6935-2

database for future reference. Output is in the form of BCP files which contain differences in two forms: a) Expected– Synthetic Breaks which are generally ignored during testing analysis. b) Unexpected- Code Breaks that signify the existing functionality getting impacted by the new code. These are the one that need to be analyzed & send over to development teams for investigation and subsequent code fix. Check points can be introduced to send over the files to a QA tester over e-mail for analysis rather than uploading onto the database.

III. Scientific Validation

After doing Adequate study of what all methodologies are available in market, Best Match Methodology was picked up, that not only eases off and simplify the Comparison of duplicate records but also wins at par from other available, in terms of its relevance to Real World Scenarios. Consideration has been put into the conversion of this methodology into an algorithm and then to subsequent implementation.

Algorithm BestMatchPair(Client Id,Test Hash,Prod Hash)
Input Client Id: Id for which bash match needs to be formed
Test Hash: Hash containing Test records for ClientId
Prod Hash: Hash containing Prod records for ClientId
Output bestMatchHash: Output Hash containing the best Match Pairs

begin bestMatchPair
1. Initialize test_length,prod_length,test_array,prod_array, big_length,small_length,ref_big_array,ref_small_array
2. Initialize temp variables i,j,k,m,best,locbest,temp,array1, array2, len1, len2
3. Split Test Hash for Client Id to retrieve the value part & store in test_array /*Step1*/
4. Split Prod Hash for Client Id to retrieve the value part & store in prod_array
5. test_length=Length(test_array),prod_length=Length(prod_array)
6. if(test_length greater than prod_length) then
7. Assign test_length to big_length,test_array to ref_big_array
8. Assign prod_length to small_length,prod_array to ref_small_array
9. else
10. Assign prod_length to big_length,prod_array to ref_big_array
11. Assign test_length to small_length,test_array to ref_small_array
12. Endif
13. for each temp i=0 covering big_length do /*Step2*/
14. Initialize best with -1 ,matchCounter with 0 & locbest with 0
15. for each temp j=0 covering small_length do
16. Pickup 0th record from ref_big_array & ref_small_array & store in k & m respectively
17. Split k into array1,len1=Length(array1)
18. Split m into array2,len2=Length(array2)
19. for each temp z=0 covering len1 do /*Step3*/
20. if(array1[z] equal to array2[z]) then
21. Increment matchCounter by 1
22. end if
23. Iterate the loop through the entire length z
24. end for
25. Assign matchCounter to temp
26. if(temp greater than best) then
27. Assign temp to best
28. Assign j to locbest
29. end if
30. Iterate the loop through the entire length j
31. end for
32. Store locbest value in bestMatchHash for key i
33. Iterate the loop through the entire length i
34. end for
35. Return bestMatchHash for Client Id
end bestMatchPair

Fig 3: Best Match Algorithm

Our Best Match Algorithm, shown in Fig3 can be applied to normal plus duplicate records but it gains most when applied in case of duplicacy scenario. Best Match Algorithm inputs 1) Client Id: Id having duplicate records 2) Test Hash: Hash having test values for Client Id 3) Prod Hash: Hash having Prod values for Client Id. The algorithm outputs the BestMatchHash thus generated for Client Id.

Proc. of the Second Intl. Conf. on Advances in Electronics, Electrical and Computer Engineering -- EEC 2013

ISBN: 978-981-07-6935-2

Step 1: Split the Test & Prod Hash into individual array elements. (lines 6-8) compares the test & prod length and takes the test array to be a big array in case if test is greater than prod. (lines 10-12) takes prod array to be big if prod is greater than test.
Step 2: Iterate Big Array over Small (lines 13-14) initializes a loop variable over the big array length and initializes few temp variables. (line 15) iterates a loop variable over small array. (lines 16-18) picks up the first record of big & small array, splits it to form sub arrays so as to store individual values in array cells.
Step 3: Finding the Best Match Pair (line 19) iterates through the temp array having test & prod values. (lines 20-22) compares the cell values one by one from the two temp arrays that signifies the test & prod value. Every time it finds a match, it increments the Counter for that ClientId. If the newly calculated matchCounter is greater than base value, the same becomes the new base value. This way for each record from big array the whole small array is scan through to calculate BestPair value .This is stored in a BestMatchHash and returned by the algorithm as an output as shown in (line 35).
This methodology at its very best simulates any real time scenario where in a person needs to find best pair among the 100 of Duplicate Records available. Below is a sample Real Time scenario

Real Time Example

Scenario: XYZ is a banking firm which deals with portfolio, holdings, transaction and performance data of its millions of customers. Each client has nearly 10-15k of trading data on a daily basis which needs to be compared and consolidated to get the day end report for audit purpose. The methodology that best suites this scenario is Best Match.

Let ABC be a client that has 111 as transaction id that has duplicate records M, N & P on the test side that needs to be compared with the corresponding record Q from Prod side. Best Match Algorithm will start by parsing M, N, P & Q record of 111 transaction id into individual cell values of a data structure already chosen for the purpose. A counter is then initiated and the first record M of 111 transaction id is compared with the record Q from opposite side. Let's say the Match counter gets the value of 5 that signifies 5 matching cell values between M & Q. Take this as the bestCount with M assumed to be best pair of Q till point. The Algorithm will pick up the next record N and calculate the best match counter in the same way. Let this counter value be 6 and represented as Count1. Algorithm will compare bestCount and Count1 and will store Count1 into bestCount with former greater than the later. Now the best pair of Q gets changed to N. This will continue till all the records are parsed through and we get a final Best Pair. This way of comparing ensures the 2 records get compared in the best possible way and as per Best Practice standards.

Formula Representation

Let $T=T_1(vaUvbUvc\ldots Uvn)\ U\ T_2(vaUvbUvc\ldots Uvn)\ U\ T_3(vaUvbUvc\ldots Uvn)\ \ldots..\ U\ T_n((v_1Uv_2Uv_3\ldots Uvn)$
$P=P_1(vxUvyUvz\ldots Uvn)\ U\ P_2(vxUvyUvz\ldots Uvn)\ U\ P_3(vxUvyUvz\ldots Uvn)\ \ldots...\ U\ P_n(v_1Uv_2Uv_3\ldots Uvn)$

$Ti \subset T = TVi \subset TV = (vaUvbUvc\ldots Uvn)$ where $i=0,1,2,3\ldots n$
$Pi \subset P = PVi \subset PV = (vxUvyUvz\ldots Uvn)$ where $i=0,1,2,3\ldots n$

$TVi \cap PVi = (vaUvbUvc\ldots Uvn) \cap (vxUvyUvz\ldots Uvn)$ where $i=0,1,2,3\ldots n$

$BM_1=va\sim vx\ \{\{\ (\sum va = \sum vx) > (\sum vb=\sum vx) > (\sum vn)\ \}U\ \{\ (\sum vx=\sum va) > (\sum vy=\sum va) > (\sum vn)\ \}\}$

$BM_2=vb\sim vz\ \{\{\ (\sum vb = \sum vz) > (\sum vb=\sum vy) > (\sum vn)\ \}U\ \{\ (\sum vz=\sum vb) > (\sum vz=\sum vc) > (\sum vn)\ \}\}$

$BMi=BM_1\ U\ BM_2\ U\ \ldots.\ BM_n$

Fig 4: Formula Representation for Best Match Algorithm

As shown in Fig 4, to find the best Match set T & P each representing Test & Prod values are first split into subsets Ti & Pi each containing individual column values which are further compared against each other in the form of TVi & PVi. A Best Pair value is only decided when a value set from Test side TVi matches with a value set from Prod side PVi and the match value is greater than all the other value sets from Ti & Pi. This is repeated for all the set values and a Best Match Hash BMi is formed. Each subset of BMi represents a best match pair of value set from set T & P.

Proc. of the Second Intl. Conf. on Advances in Electronics, Electrical and Computer Engineering -- EEC 2013

ISBN: 978-981-07-6935-2

IV. Data Structure

Implementation of the methodology described above requires a data structure that can best serve its purpose. The data structure should be Easy to Implement, Easy to Debug, Easy to Understand, Flexible for any generic scenario. Along with that it should be as per the best practice industry standards. One such Data Structure is Hash Structure. Hash Structure wins from the other available structure like Arrays, Vectors by allowing the record values to be saved in the form of Key-Value Pair where key is a unique among all the records and the value is the remaining cell values[5]. Hashing when combined with Best Match methodology simplifies the processing to find the Best Match Pair. Comparing cell values stored in Hash structure is a lot simpler and industry compliant. This simplicity further adds to the value that the Best Match brings to the table. Combination of both gives the Automation Framework huge gains and helps to achieve the overall goal of Comparison Automation in the best possible way.

V. Framework Implementation

Perl programming language was used for the implementation of the Automation Framework, architecture of which is explained above[7]. It consists of 250 Lines of Code and is compatible across different operating systems like Windows, Unix/Linux.

Integrated Development Environment (IDE) named Padre is used for the development work of the Automation Framework. This framework can easily be used across different disciplines/fields of an IT industry with changes required only in configuration files. The framework not only reduces the human effort, reduces the overall cycle time, reduces the % of errors creeping in, almost no manual intervention involved but also achieves the end-to-end Regression Testing Automation. When seen in IT terms, the comparison framework if used will save a lot of dollar money that is the main goal of financial banks across the globe especially during the hard recession times or global turmoil.

Framework Output

Below is a sample output as generated by the code:

```
patrick@ms.com : Transaction : gainloss_NA_USD.out [Error]

        Key:32-95702    COLUMN trans_transaction_date: (pVal) 12/15/2009 (tVal) 14/15/2009

        Key:32-95702    COLUMN trans_security_description: (pVal) AOL INC<BR>AOL INC CASH IN LIEU .999994SHRS<BR>AT $23.860000 AOL INC (tVal) AOL INC<
GC CASH IN LIEU .999994SHRS<BR>AT $23.860000 AOL INC

patrick@ms.com :Transaction :  transactions_NA_USD.out -> OK

patrick@ms.com : Performance : performance_NA_USD.out [Error]

        Extra Section Found:
        32-95701:-7.44:-5.93:02/19/2010:03/31/2001:50138:197051.91:146913.91:6404.95:286501.79:17474.34:-139587.88:-11069.39:SPXDIV:1.40:-0.24:S & P
X CLOSING PRICE TR:DJIA:-0.25:DOW JONES INDUSTRIAL AVERAGE PR:-49.70:13.19:5.30:patrick@ms.com:USD

=====================================
Total number of failed clients -> 4

FOR CLIENT -> jsaxe@ms.com
        FAILED QUERY -> Holdings
        FAILED QUERY -> Holdings

FOR CLIENT -> fon@ms.com
        FAILED QUERY -> Performance
        FAILED QUERY -> Performance

FOR CLIENT -> mlporter@ix.netcom.com -> Not a Valid User [Error]


FOR CLIENT -> patrick@ms.com
        FAILED QUERY -> Transaction
        FAILED QUERY -> Performance
=====================================
```

Fig 5: Sample Output of Comparison Automated Framework

ISBN: 978-981-07-6935-2

Differences as displayed are categorized into 2 types: 1.Individual Column Value Differences 2.Extra Section Differences.

This framework can easily be enhanced for any field or scenario. With move of time and with inflow of new requirements, the design can easily be extended & reused.

Acknowledgment

This research paper is made possible with due help & guidance from colleagues & lecturers from Lingayas University. No monetary help has been sought for this purpose from anybody. Difficulties faced while preparing the paper was sorted out mutually among the 2 authors. The journey through the paper was full of learning that helped us uncover various hidden facts. Many anonymous reviewers provided valuable suggestions that helped in the implementation & presentation of test summary results. We will also like to thank our parents who stayed with us during the course and kept on supporting till the end. This couldn't have been possible without them.

References

[1] Hanna Remmel, Barbara Paech, Peter Bastian & Christian Engwer,"System Testing a Scientific Framework Using a Regression-Test Environment",IEEE CS,pp. 40-41,2012

[2] B.Kitchenham,L.Pickard, and S.Pfleeger,"Case Studies for Method and Tool Evaluation",IEEE Software,vol.11,no.4,pp-52- 62,July 1995

[3] Hanna Remmel, Barbara Paech, Peter Bastian & Christian Engwer,"System Testing a Scientific Framework Using a Regression-Test Environment",IEEE CS,pp. 42,2012

[4] Mary Jean Harrold,"Reduce, Reuse, Recycle, Recover: Techniques for Improved Regression Testing".IEEE CS,,vOL-28,pp. 1,2009

[5] S . Elbaum, A.G. Malishevsky and G. Rothermel,"Test Case Prioritization: A Family of Empirical Studies".IEEE CS,vOL-28,pp. 2,2002

[6] Hanna Remmel, Barbara Paech, Peter Bastian & Christian Engwer,"System Testing a Scientific Framework Using a Regression-Test Environment",IEEE CS,pp. 39,2012

[7] Mark-Jason Dominus,"Perl:Not Just for Web Programming",IEEE Software,Jan-Feb 1998

[8] Macario Polo, Pedro Reales, Mario Piattini, and Christof Ebert,"Test Automation",IEEE Software,pp 84-89,Jan-Feb201